|  |  |
| --- | --- |
| **Mark** | **/11** |

|  |  |  |  |
| --- | --- | --- | --- |
| Team name: | *A5* | | |
| Homework number: | *HOMEWORK 05* | | |
| Due date: | 20/10/2024 | | |
|  |  |  |  |
| Contribution | NO | Partial | Full |
| Alessio Spineto |  |  | *x* |
| Riccardo Lamarca |  |  | *x* |
| Sofia Cecchetto |  |  | *x* |
| Annamaria De Togni |  |  | *x* |
| Emma Crespi |  |  | *x* |
| Notes: none | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Project name | ADC + LCD UART string | | |
| Not done | Partially done  (major problems) | Partially done  (minor problems) | Completed |
|  |  |  |  |
| **Project 2b:** acquire the potentiometer voltage using a timer to trigger a conversion at a regular conversion rate of 1 Hz and sending the value to a remote terminal.    We enabled the potentiometer (PA1) analog input channel as ADC1\_IN1 and also enabled GPIO\_Analog since we are reading an analog signal. This saves resources since we are not using the digital buffer, not needed in this case.    The two USART pins are enabled by default. In the connectivity section of the GUI, we selected USART2, opening its ‘Mode and Configuration’ page.  There, in the DMA settings, we added a new DMA request, selecting the transfer mode USART2\_TX with high priority.  In the same window, under Parameter Settings, we selected a 115200 baud rate and 8 bit word length.    We configured the ADC by enabling its global interrupt. We set the conversion to be triggered by *Timer 2 Trigger Out event* and *Trigger detection on the rising edge,* and the sampling time at 480 cycles.    We set TIM2 to send an update event every second, by setting the prescaler to 8399 and the ARR to 9999.      Inside the *main.c* file we initialize both the ADC and the timer.    Once the ADC conversion is complete, it sends an interrupt.  We converted the acquired digital value into the voltage range [0 - 3,3]V and then sent the string to the remote terminal.  We also made sure to enable the use of *float* with printf in the C build settings, to correctly display the results. | | | |
| **Project 2c:** acquire the potentiometer voltage using a timer to trigger a conversion at a regular conversion rate of 1 Hz and showing the value on the LCD.  We configured the pinout to use the LCD display by setting PA4, PB1, PB2, and PB12-15 to GPIO\_Output. The “PMDB16\_LCD.h” library that we import manages the use of the LCD monitor through these pins. The ADC and TIM2 settings were left unchanged compared to the first part of the homework.    Moving on to the coding part, we included the LCD, math.h and string.h libraries to access some utilities, like *snprintf()*.    We then initialized the ADC, the timer and the LCD.    Then we defined the callback for the ADC. In this part of the code, we read the value converted by the ADC and scaled it to be in the range [0 - 3.3]. The converted value is then inserted in a string of fixed length, and finally printed on the LCD’s first row.  Additionally, we created another variable called *value\_bar* containing the ADC’s values converted in the range [0 - 80]. Then we called *lcd\_drawBar()* which, given an integer value between 0 and 80 (*value\_bar*), prints a bar graph on the second row of the LCD. | | | |
| **Project**: Try sending from the PC via UART a string of variable length that is displayed on the LCD.  As in the above projects, we configured the pins to use the LCD display. Also, the configuration of USART remains the same.  In the *main.c* file we included the following libraries.  Then, we initialized the LDC and turned on the backlight. We started UART reception in DMA mode.    In the first part of our code, we defined the variables used to store the received strings and print on the LCD.  The *print\_string* function exploits the use of the *c* function *strncpy* to divide and copy inside the variables buff\_h and buff\_l , the values stored inside buff\_str. Then, these values are printed on the LCD’s first and second row. We also need to clear the values stored in buff\_str in case of the arrival of a new string.  In the last part of the code we set the properties of the UART Receiver Callback. First, we store the values received by UART inside the variable buff. For each incoming character, different from a terminating one, we stored its value inside the variable buff\_str. If the letters sent are more than 32, we display the first 32 characters by calling the function *print\_string().*  When the user sends a terminating character (for example using by typing ENTER), we display the entire sentence on the LCD by calling *print\_string().* | | | |
| Professor comments: | | | |